Electronic Version
Stylesheet Version v1.1.1

Description

SYSTEM, METHOD AND STORAGE MEDIUM FOR MEMORY MANAGEMENT

BACKGROUND OF INVENTION

[0001] The invention relates to memory management and in particular, to a tag controlled prefetch buffer management system that is part of a cache system.

[0002] In processing systems such as computers, the data to be utilized by a processor is stored in a memory (e.g., main memory, lower level memory) and control logic manages the transfer of data between the memory and the processor in response to requests issued by the processor. The data stored in the main memory generally includes both instructions to be executed by the processor and data to be operated on by the processor. For simplicity, both instructions and true data are referred to collectively herein as "data" unless the context requires otherwise. The time taken by a main memory access is relatively long in relation to the operating speeds of modern processors. To

address this, a cache memory with a shorter access time is generally interposed between the main memory and the processor, and the control logic manages the storage of data retrieved from the main memory in the cache and the supply of data from the cache to the processor.

[0003] A typical cache is organized into multiple "lines", each line providing storage for a line of data from the main memory which may be many bytes in length. When the processor issues a request for data contained in a particular line in a page, or block, the control logic determines whether that line is stored in the cache. If the line is stored in cache (i.e., there is a cache hit), the data is retrieved from the cache. If the line is not stored in cache (i.e., there is a cache miss), the data must be retrieved from the main memory and the processor is stalled while this operation takes place. Since a cache access is much faster than a lower level memory access, it is clearly desirable to manage the system so as to achieve a high ratio of cache hits to cache misses.

[0004] Memory latency is becoming an increasingly important factor in computer system performance. An implication of this increasing importance is that cache faults from the slowest on-chip cache are becoming more expensive in

terms of performance. One approach to mitigating this problem is to increase the size of the cache. Increasing the size of the cache may improve performance, however, cache memory is expensive in comparison to the slower, lower level memory. It is therefore important to use cache memory space as efficiently as possible.

[0005] One way to improve the efficiency of a cache memory system and to decrease memory latency time is to attempt to anticipate processor requests and retrieve lines of data from the memory in advance. This technique is known as prefetching. Prefetching can be performed by noting dynamic properties of the reference data stream such as sequential and/or stride access. Alternatively, prefetching can be performed on the basis of stored information. This stored information might be related to patterns of access within or between pages, or to hints produced by the compiler and/or programmer.

[0006] In cache structures with prefetching, a common approach is to have a prefetch buffer which holds lines that have been prefetched. Having such a separate buffer avoids pollution of the cache proper due to mistaken prefetches. However, it is often difficult to coordinate the contents of such a prefetch buffer with logic that determines what to

prefetch as a function of ongoing accesses, or stored information. In addition, searching a prefetch buffer may require multiple associative lookups for a single operation.

SUMMARY OF INVENTION

[0007] One aspect of the invention is a system for memory management. The system includes a tag controlled buffer in communication with a memory device. The memory device includes a plurality of pages divided into a plurality of individually addressable lines. The tag controlled buffer includes a prefetch buffer including at least one of the individually addressable lines from the memory device. The tag controlled buffer also includes a tag cache in communication with the prefetch buffer. The tag cache includes a plurality of tags, where each tag is associated with one of the pages in the memory device and each tag includes a pointer to at least one of the lines in the prefetch buffer. Access to the lines in the prefetch buffer is controlled by the tag cache.

[0008] Another aspect of the invention is a system for memory management including a random access memory. The random access memory includes at least one of the lines. Each line is associated with a page in a memory device

and space in the random access memory is allocated on per line basis. The system also includes a first cache device with a plurality of tags. Each tag corresponds to one of the pages in the memory device and each tag indicates the location in the random access memory of the at least one line associated with the tag.

[0009] Another aspect of the invention is a method for memory management. The method includes receiving a fault notification from a first cache device. The fault notification includes a fault page identifier and a fault line identifier. A second cache device is accessed to determine if a line in a random access memory corresponds to the fault page identifier and the fault line identifier. The random access memory includes at least one line associated with a page in a memory device. The second cache device includes a plurality of tags each corresponding to one of the pages in the memory device and each tag indicates the location in the random access memory of the at least one line associated with the page corresponding to the tag. The line corresponding to the fault page identifier and the fault line identifier from the random access memory is transmitted to the first cache device in response to the accessing resulting in locating the line corresponding to the

fault page identifier and the fault line identifier in the random access memory. The tag corresponding to the fault page identifier is updated in the second cache device to reflect the transmitting.

[0010] Another aspect of the invention is a method for memory management. The method includes receiving a fault notification from a requestor. The fault notification includes a fault page identifier and a fault line identifier. The method also includes determining if a tag corresponding to the fault page identifier is located in a tag cache. The tag cache includes a plurality of tags, each tag includes at least one pointer to lines in a prefetch buffer and the tag cache designates at least one of the prefetch lines. In response to locating the tag corresponding to the fault page identifier, a line corresponding to the fault line identifier is transmitted to the requestor and the tag corresponding to the fault page identifier is updated to reflect the transmitting. In response to not locating the tag corresponding to the fault page identifier, a new tag corresponding to the fault page identifier is inserted into the tag cache. In addition, the line corresponding to the fault line identifier is transmitted to the requestor and the prefetch lines included in the new tag are inserted into the prefetch buffer

via the tag cache.

[0011] A further aspect of the invention is a computer program product for memory management. The computer program product includes a storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method that includes receiving a fault notification from a requestor. The fault notification includes a fault page identifier and a fault line identifier. The method also includes determining if a tag corresponding to the fault page identifier is located in a tag cache. The tag cache includes a plurality of tags, each tag includes at least one pointer to lines in a prefetch buffer and the tag cache designates at least one prefetch line. In response to locating the tag corresponding to the fault page identifier, a line corresponding to the fault line identifier is transmitted to the requestor and the tag corresponding to the fault page identifier is updated to reflect the transmitting. In response to not locating the tag corresponding to the fault page identifier, a new tag corresponding to the fault page identifier is inserted into the tag cache. In addition, the line corresponding to the fault line identifier is transmitted to the requestor and the prefetch lines included in the new tag are inserted into the

prefetch buffer via the tag cache.

Brief Description of Drawings

[0012] Referring now to the drawings wherein like elements are numbered alike in the several FIGURES:

[0013] FIG. 1 is a block diagram of a cache system that includes a prefetch buffer and a tag cache in accordance with an exemplary embodiment of the present invention;

[0014] FIG. 2 depicts the contents of tags in an exemplary embodiment of the present invention;

[0015] FIG. 3 is a flow diagram of a method for performing prefetching in an exemplary embodiment of the present invention;

[0016] FIG. 4 is a flow diagram of replacement logic for the tag cache that may be utilized by an exemplary embodiment of the present invention; and

[0017] FIG. 5 is a block diagram of an exemplary embodiment of storage allocation in the prefetch buffer.

Detailed Description

[0018] An exemplary embodiment of the present invention includes a prefetch buffer with contents that are controlled by tags. Each tag corresponds to one page in memory. The tag includes access histories for lines in the page, or

between the page corresponding to the tag and other pages. To determine if a particular line from a page is in the prefetch buffer, a search is performed using the tags. The tags include pointers to locations in the prefetch buffer (e.g., a shared memory pool) for lines that are stored in the prefetch buffer. Tags also hold relevant reference information to aid in future prefetching. The tags control the presence and/or absence of lines in the prefetch buffer and all access to the lines in the prefetch buffer is via the tags. An advantage to this design is that all searches are centered on the structure holding the tags, thereby avoiding multiple associative lookups for a single operation. Further, the design may help to insure that recently entered tags are guaranteed space for their prefetched lines.

[0019] An exemplary embodiment of the present invention includes a memory system with a level two (L2) cache, a prefetch buffer, and a tag cache containing recently referenced tags. Each tag in the tag cache is associated with an individual page in main or lower level memory. In an exemplary embodiment of the present invention, each individual page within the lower level memory is four-thousand (4K) bytes. Tags are held in memory and are ac-

cessed on references to the prefetch buffer. Searches for tags associated with a given page in lower level memory are performed in any manner known in the art for cache memory (e.g., by associative search within a congruence class). In an exemplary embodiment of the present invention, it is assumed, for simplicity, that the tag cache is fully associative. The tags include information used in prefetch or replacement decisions. In an exemplary embodiment of the present invention, the tags include information about what lines should be fetched from a page, given a cache fault occurring on one of its lines, as well as some flag bits. The prefetch buffer, along with the tag cache is referred to collectively as a tag-controlled buffer (TCB).

[0020] Certain events (e.g., a reference to one of the lines in a page) may cause a page tag to be fetched (e.g., from a page table) and placed in the tag cache, possibly displacing one or more other tags. The information held in the tag is used to prefetch lines from the page corresponding to the tag, as well as possibly tags from other pages. Fetched lines are placed in a shared prefetch buffer that may be implemented by a shared memory pool that resides on the processor integrated circuit. The prefetched

lines associated with a tag are placed in the prefetch buffer, at places determined by a free space list that is maintained by hardware and/or software. If there is insufficient space for the prefetched lines, other tags are deleted from the tag cache and written back to memory so as to free sufficient space. In addition, the lines in the prefetch buffer that correspond to the deleted tags are removed from the prefetch buffer. As lines are referenced (i.e., to satisfy L2 cache faults), these lines may be deleted from the buffer and placed in the L2 cache. Space occupied by such lines in the prefetch buffer is then placed on the free space list. Displaced tags are updated and written back to memory.

[0021] FIG. 1 is a block diagram of an exemplary cache system that includes a L2 cache 106, a lower level memory 108, a prefetch buffer 104, a tag cache 102 and a processor 114 in accordance with an exemplary embodiment of the present invention. For purposes of illustration, it is assumed that the lower level memory 108 is divided into pages, each of 4K bytes. In addition, cache lines are assumed to contain one hundred and twenty-eight (128) bytes, so that there are thirty-two (32) lines per page. The tag-controlled buffer (TCB) 110 includes the tag cache

102 and the prefetch buffer 104. As shown in FIG. 1, the TCB 110 is in communication with both the cache 106 and the lower level memory 108 to allow the prefetch buffer to transfer lines into the cache 106, to read data from the lower level memory, and to read/write tags into the lower level memory 108. In addition, the cache 106 is in communication with the lower level memory 108 to move data between the two. The TCB 110 is also in communication with the processor 114. The processor 114 includes instructions to implement the processes described herein and may be physically located in a variety of devices (e.g., on the TCB 110, on a memory controller) that are accessible by the TCB 110. Also, as is known in the art the instructions may be physically located on one or more processors 114.

[0022] The tag cache 102 depicted in FIG. 1 includes recently referenced TCB tags 112 which include pointers to one or more locations in the prefetch buffer 104. In addition, the TCB tags 112 may include other information related to lines held in the prefetch buffer 104. In an exemplary embodiment of the present invention, the tag cache 102 is organized as a standard cache structure with the storage of TCB tags 112 divided into a set of equivalence classes,

where the TCB tags 112 are searched for by equivalence class. Alternative cache structures known in the art may be implemented by the tag cache 102. In an exemplary embodiment of the present invention, access to the contents of the prefetch buffer 104 is only via the tag cache 102.

[0023] FIG. 2 depicts the contents of tags in an exemplary embodiment of the present invention. The TCB tags 112 are stored in the tag cache 102 and the memory tags 216 are stored in lower level memory 108. An exemplary embodiment of a TCB tag 112 includes a page identification field 202, a presence bits field 204, a history vector field 206, a pointers field 208 and a flags field 210. The page identification field 202 includes the address of the page in lower level memory 108 (i.e., the real address). The presence bits field 204 includes one bit for each line in the page. As described previously, for pages that that are 4K, with lines of one hundred and twenty-eight (128) bytes, there are thirty-two (32) lines per page. Therefore, there are thirty-two (32) presence bits included in the presence bits field 204, each corresponding to a different line in the page. The presence bit is set to "1" if the corresponding line is currently included in the prefetch buffer 104 and the

presence bit is set to "0" if the corresponding line is not currently included in the prefetch buffer 104.

[0024] The history vector field 206 in the TCB tag 112 includes a collection of information regarding how lines in the page have been referenced in the past. As the TCB tag 112 is accessed and lines are referenced, this field may be updated and used to assist in making a determination about what data should be included in the prefetch buffer 104. In an exemplary embodiment of the present invention, the history vector field 206 is stored as a history sequence or vector containing the thirty-two (32) most recent line references from the page. The pointers field 208 contains data that indicates the location of the lines in the page that are currently located in the prefetch buffer 104. The pointers field 208 includes a pointer to a prefetch buffer location corresponding to each line from the page that is currently located in the prefetch buffer 104, as determined by the value of the presence bits field 204. The flags field 210 includes spare bits that my be utilized for other purposes, such as the status of the page.

[0025] When the TCB tag 112 is ejected from the tag cache 102, a subset of the TCB tag 112 is stored back into lower level memory 108 as a memory tag 216. The memory tag 216

depicted in FIG. 2 includes the page identification field 202, a reference history field 212 and a flags field 214. The page identification field 202 is the same field discussed previously in reference to the TCB tag 112 and it includes the real address. In an exemplary embodiment of the present invention, the reference history field 212 includes the same data as the history vector field 206 in the TCB tag 112. In alternate exemplary embodiments, the reference history field 212 in the memory tag 216 includes a subset and/or additional data than the history vector field 206 in the TCB tag 112. The data in the reference history field 212 may be utilized to determine which lines from a page should be prefetched when a tag corresponding to the page is entered into the tag cache 202. The flags field 214 contains spare bits that may be utilized for other purposes. The TCB tag 112 and memory tag 216 described in reference to FIG. 2 are examples of one method of implementation. As is known in the art they may be modified by adding and/or removing fields without departing from the spirit of the invention.

[0026] FIG. 3 is a flow diagram of a method for performing prefetching in an exemplary embodiment of the present invention. At step 302, a cache fault occurs as a result of

a reference to a line not currently in the cache 106. When the cache fault occurs, the faulted line is fetched either from the prefetch buffer 104 or from lower level memory 108. At step 304, a check is made to determine if a TCB tag 112 associated with the faulted line is currently in the tag cache 102. This is performed by accessing the tag cache 102 and searching the page identification fields 202 for a page identification that matches the page identification associated with the faulted line. As described previously, the searching is performed using standard cache methods such as performing an associative search within a congruence class.

[0027] If a TCB tag 112 corresponding to the page where the faulted line is stored in lower level memory is located in the tag cache 102, then step 306 is performed to fetch the faulted line into the cache 106. The presence bits field 204 is examined for the value of the bit corresponding to the faulted line. If the bit value indicates that the faulted line is located in the prefetch buffer, then the line is moved into to the cache 106 and deleted from the prefetch buffer. The line is effectively removed by changing the bit value corresponding to the line to "0" because access to the prefetch buffer 104 is through the tag cache

102. In addition, the value of the pointer corresponding to the faulted line in the pointers field 208 may be deleted in order to delete the faulted line from the prefetch buffer. Alternatively, the bit value associated with the faulted line may indicate that the faulted line is not currently located in the prefetch buffer (e.g., the bit value has a value of "0"). In this case, the faulted line is fetched directly into the cache 106 from the lower level memory. Once the line has been fetched into the cache 106, processing ends at step 312.

[0028] If a TCB tag 112 corresponding to the page where the faulted line is stored in lower level memory is not located in the tag cache 102, as determined at step 304, then step 308 is performed to fetch the memory tag 216 associated with the faulted line into the tag cache 102 and to create a TCB tag 112 associated with the page containing the faulted line. The memory tag 216 may be located in lower level memory 108 or in the cache 106. Once a TCB tag 112 associated with the faulted line is added to the tag cache 102, step 306 is performed as described previously, by fetching the faulted line directly into the cache 106 from the lower level memory. Once the line has been fetched into the cache 106, processing ends at step 312.

In an alternate exemplary embodiment, the order of performing steps 306 and 308 is reversed, resulting in fetching the line into the cache 106 first and then creating the TCB tag 112 in the tag cache 102. In another alternate exemplary embodiment, steps 306 and 308 may be executed concurrently.

[0029] In addition, to step 306, once step 308 has been performed, step 310 is executed to fetch lines associated with the faulted line into the prefetch buffer 104, if the associated lines are not already contained in the cache 106 or the prefetch buffer 104. In an exemplary embodiment of the present invention, the associated lines are those that have entries in the history vector field 206 of the TCB tag 112. Any algorithms known in the art for determining lines to prefetch may be implemented by an exemplary embodiment of the present invention. This may result in additional TCB tags 112 being entered into the tag cache 102 and/or additional lines being entered into the prefetch buffer 104. In step 310, the presence bits field 204 of the TCB tag 112 is updated to reflect the corresponding lines contained in the prefetch buffer 104. When step 310 is completed, processing ends at step 312.

[0030] FIG. 4 is a flow diagram of replacement logic for the tag

cache that may be utilized by an exemplary embodiment of the present invention. For simplicity in describing the cache replacement logic, a least recently used (LRU) replacement algorithm for the tag cache 102 is assumed. Other cache replacement logic as is known in the art may also be utilized by exemplary embodiments of the present invention. For the LRU replacement logic, the TCB tags 112 are ordered according to how recently they have been referenced. Here a reference includes an action which reads or modifies the TCB tag 112. At step 402, a current TCB tag 112 is created by fetching and augmenting a memory tag 216. The current TCB tag 112 is inserted into the tag cache 102 in the most-recently-referenced position in the ordering. Alternatively, at step 402, a current TCB tag 112 is created by reading or modifying a TCB tag 112 already located in the tag cache 102.

[0031] At step 404, a determination is made about whether there is adequate space in the tag cache 102 for the current TCB tag 112. If there is adequate space in the tag cache 102 then step 406 is performed to insert the current TCB tag 112 into the tag cache 102 if it does not already exist in the tag cache 102 and processing continues at step 412. Alternatively, if additional space is required in the tag

cache 102, then step 410 is performed and the least recently referenced TCB tag 112 is deleted from the tag cache 102. This deleting is performed by updating the TCB tag 112 to create the memory tag 216 and writing the memory tag 216 back to lower level memory 108. In addition, lines from this page referenced by the TCB tag 112 that are currently held in the prefetch buffer are then added to the free space list. In an exemplary embodiment of the present invention, the memory tag 216 as written back includes a page identifier in the page identifier field 202, a reference history vector of the thirty-two (32) most recently referenced lines from this page in the reference history vector field 212, as well as the above mentioned flag bit fields 214.

[0032] Next, at step 412, a check is made to determine if there is adequate space for the currently prefetched lines associated with the current TCB tag 112 in the prefetech buffer 104. If there is enough space for the currently prefetched lines then step 414 is performed to insert the currently prefetched lines into the prefetch buffer 104. As described previously, this is performed by adding the currently prefetched lines to the buffer and updating the corresponding TCB tag 112 to signify the presence (via set-

ting the corresponding bits in the presence bits field 204 to a "1") and location (via updating data in the pointer field 208) of the new lines in the prefetch buffer 104. Processing then ends at step 408. Alternatively, if it is determined, at step 412, that there is not adequate space for the currently prefetched lines associated with the current TCB tag 112 then step 416 is performed to free up space in the prefetch buffer 104. At step 416, TCB tags 112 and associated lines in the prefetch buffer 104 are deleted until there is enough free space in the prefetch buffer 104 for the currently prefeteched lines associated with the current TCB tag 112. This may be performed using a LRU algorithm. Processing then continues to step 414 as described previously.

[0033] FIG. 5 is a block diagram of an exemplary embodiment of storage allocation in the prefetch buffer 104. FIG. 5 illustrates the operation and use of a free space list 502 that contains a list of the available locations for holding cache lines in the prefetch buffer 104. As space is freed up as indicated above, via the deletion of entries in the tag cache 102, the location used by the corresponding lines are added to the free space list 502. As space is allocated, the locations of the added space are deleted from the free

space list 502. Any data structures and/or methods, both hardware and/or software, known in the art may be utilized to implement the free space list 502. For example, the free space list 502 may be implemented by a linked list and associated logic.

[0034] Exemplary embodiments of the present invention may be utilized with systems that include other sizes of pages, lines, tags and cache than those specifically discussed above. In addition, exemplary embodiments of the present invention are not limited to particular levels of cache (e.g., L2 as described previously) and may be applied to any level of a storage hierarchy.

[0035] An exemplary embodiment of the present invention includes having the location of prefetched lines in a prefetch buffer being designated by pointers in tag entries in a tag cache. An advantage to this design is that all searches are centered on the structure holding the tags, thereby avoiding multiple associative lookups for a single operation. This results in only a single search being required when attempting to access a cache line which may be in the prefetch buffer, along with whatever other lines from the same page are resident in the prefetch buffer, despite the fact that both the associated tag as well as the

cache line need to be found. In addition, when a tag is flushed from the tag cache and written back to memory, the space for its associated lines in the prefetch buffer is freed. This may help to insure that recently entered tags are guaranteed space for their associated prefetched lines.

[0036] As described above, the embodiments of the invention may be embodied in the form of computer implemented processes and apparatuses for practicing those processes. Embodiments of the invention may also be embodied in the form of computer program code containing instructions embodied in tangible media, such as floppy diskettes, CD-ROMs, hard drives, or any other computer readable storage medium, wherein, when the computer program code is loaded into and executed by a computer, the computer becomes an apparatus for practicing the invention. An embodiment of the present invention can also be embodied in the form of computer program code, for example, whether stored in a storage medium, loaded into and/or executed by a computer, or transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via electromagnetic radiation, wherein, when the computer program code is loaded into and executed by a computer, the computer becomes

an apparatus for practicing the invention. When implemented on a general-purpose microprocessor, the computer program code segments configure the microprocessor to create specific logic circuits.

[0037] While the invention has been described with reference to exemplary embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the scope of the invention. In addition, many modifications may be made to adapt a particular situation or material to the teachings of the invention without departing from the essential scope thereof. Therefore, it is intended that the invention not be limited to the particular embodiment disclosed as the best mode contemplated for carrying out this invention, but that the invention will include all embodiments falling within the scope of the appended claims. Moreover, the use of the terms first, second, etc. do not denote any order or importance, but rather the terms first, second, etc. are used to distinguish one element from another.